Hands-on session

The goal of this session is to gather experiences with some tools, and learn a bit about where they excel, and where other tools are better suite. For every task, there is a tool that is best for it. There will not be any evaluation of your results, but I want you to spend time with the tools. So make good use of this time and do something that's fun. Try to find a good mix between finding attacks and understanding the flaws by fixing attacks. Here is the assignment, the tasks are described below.

1. Task: authentication protocol
   - ProVerif Siavash
   - CryptoVerif Rati
   - ProScript Matthias
2. Task: Hardware security module
   - SAPIC Sven
   - AIF-omega Alex
   - StatVerif (not assigned)
3. Task: Simple messaging protocol with DH
   - tamarin-prover Hizbullah
4. Task: Port examples from one to the other or make up protocol providing unlinkability.
   - APTE Jayanth
   - Akiss (not assigned)

Authentication protocol

Start of by formalising any authentication known to work. There is database of those called Spore. Some simple ones to start with are are Needham-Schroeder-Lowe, Wide-mouth-frog and Denning-Sacco. Leave out a couple of terms in messages, you should discover attacks. Can you rediscover known attacks?

You should get an idea of what works or does not work quickly. Now try inventing your own protocol, maybe you can come up with one? If you are super ambitious: I've always wondered if Firefox's synch mechanism is secure. Here is the reference.

Note that ProVerif and CryptoVerif have a rich database of examples to get started with. ProVerif does have two versions of its calculus, one where all terms are of the same type, and one where keys and certain symbols can have types and can thus not be confused by the protocol. This often simplifies verification, but now some examples are for the untyped calculus, and some for the typed one. ProScript is pretty new, but it translates to ProVerif, so you can try implementing example protocols from ProVerif and re-gain them by compilation to get started.

The goal is to show authentication, and, if applicable, secrecy of the key-exchanged. For CryptoVerif or ProVerif in diff-mode, even strong secrecy (indistinguishability) can be shown.

Hardware security module (HSM)

All three tools were designed to deal with stateful processes, so protocol verification can be used (mis-used?) to verify the logical interface and secure key-storage device exposes. We assume the adversary has gotten hold of the server connected to the HSM and thus there is only one honest party in the protocol: the HSM itself.

Start of with the following model, and then add new things, refine, or come up with something on your own. The honest party has at least three three processes, which can all be queried repeatedly :

1. create: on receiving a request, creates a keys k with with running number n and stores (k, n).
2. encrypt: given a query (n_1, n_2) encrypt k_1 with k_2, but only if k_2 was created earlier than k_1, i.e., $n_2 < n_1$
3. corrupt: given a query n give out the corresponding k

The goal is to show that each key k created with level n is secret, unless the adversary corrupted a key created earlier, i.e., one with $n' > n$.

Messaging protocol

This is similar to the first task, but you should use Diffie-Hellman exponentiation, which is tamarin's strength. Start of with signed Diffie-Hellman. Observe the interesting security property, called perfect forward secrecy. What happens if you leave out the signatures? Try to come up with a similar way of adding authenticity to the famour Diffie-Hellman key-exchange protocol.

Unlinkability

Both utilities provide examples for unlinkability and secrecy in key-exchange. Study how these notions make sense. Now you can start by porting examples from Akiss to APTE, but you can also try to design your own protocol. If you have an intuition for unlinkability in the passport example, try to modify the protocol, see what brakes, and try to repair it in a different way.